Simulation Modelling Practice and Theory 58 (2015) 172–187

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

On optimization of simulation execution on Amazon EC2 spot market

Bogumił Kamiński *, Przemysław Szufel

Warsaw School of Economics, Al. Niepodległości 162, 02-554 Warszawa, Poland

ARTICLE INFO

Article history:
Available online 10 June 2015

Keywords:
Cloud based simulation
Simulation of the cloud
Cost optimization
Amazon EC2
Spot prices

ABSTRACT

Large scale simulations require considerable amounts of computing power and often cloud services are utilized to perform them. In such settings the execution costs can be significantly decreased through the use of the Amazon spot price market. Its downside is that Amazon can interrupt the user's computations when her bid price is too low. This poses a problem in finding an on-line bidding algorithm that balances the computation cost and the simulation experiment completion time.

We identify key drivers governing the spot prices on Amazon EC2 and using these insights propose an adaptive bidding strategy that simultaneously minimizes the computation cost and the delays due to computation termination. We show that bidding close to a spot price and dynamically switching between instances is a strategy that is efficient and simple to implement in practice.

In the paper we present a simulator of the EC2 spot pricing mechanism. The simulator can be easily used to develop and test other bidding strategies on Amazon spot price market.

1. Introduction

The goal of this paper is to propose an algorithm for a cost and time optimization for running simulations on public computational clusters with a spot pricing mechanism. The algorithm is implemented in Python and is ready for application to real-life computationally intensive simulations executed on the Amazon Elastic Compute Cloud (Amazon EC2).

Amazon is the largest cloud computing provider and offers many server types in eight regions around the globe. Amazon offers its customer three pricing mechanisms: on-demand, reserved instances and spot pricing. In this paper we focus on the spot pricing mechanism. It offers the lowest prices on average at the expense of the risk of an abrupt termination of computations when the user's bid price becomes lower than the current spot price. For applications where the up-time is crucial this is a severe limitation. However, when running simulations one can decide to allow for breaks in computation if the cost reduction is significant enough. Amazon is fully aware of this situation and EC2 documentation [1] discusses four main state-of-the-art architectures that are designed for scientific computing on spot instances.

However, customers wanting to take full advantage of spot pricing face a complex ecosystem. Firstly, the spot prices are volatile and there is a possibility of a simulation termination. In such a situation, if the computation state is not saved (i.e. the simulation has not been check-pointed) the results are lost. Secondly, there are over 100 different server types to choose

* Corresponding author. Tel.: +48 (22) 564 60 00.
E-mail address: bkamins@sgh.waw.pl (B. Kamiński).

http://dx.doi.org/10.1016/j.simpat.2015.05.008
1569-190X/

from, but, on the other hand, the number of spot instances that a user can initiate in parallel is limited and controlled by Amazon. Thirdly, when considering available computation power one has to take into account the booting time of the virtual machine – see [2] for analysis of dependencies between instance booting time and instance type. Finally, the Amazon EC2 spot billing mechanism has several nuances that can be exploited to improve the cost-performance of computations. Due to this complexity the spot pricing analysis attracts a lot of attention in the literature.

Researchers consider the spot market for computing power from a provider's perspective as well as from a user perspective (a detailed literature review is provided in Section 2; here we give its summary). Analysis of the cloud provider's side focuses on designing a pricing mechanism that maximizes profits, see for example [3]. Other papers also include market design postulates. For instance Vanmachlen et al. [4] propose an extension of the existing spot market with the capability of bidding for future prices for computing power (futures market).

The problem of the computational grid pricing from the end-user's perspective focuses on cost-reduction. Several papers discuss how cloud utilization is becoming an important approach in cost-optimizing for large scale scientific computing – an overview of cloud scientific computing applications is given in [5]. Mattess et al. [6] point out that using cloud computing and spot instances is an efficient cost management technique for peak loads in local computational clusters. Efficient utilization of spot instances requires a user to find the optimal bidding strategy. Javadi et al. [7] analyze hourly and weekly patterns in spot price and time between price changes and they propose a mixture of Gaussian distributions to model the price patterns. In [8] a model of bidding strategies with service level agreement constraints is presented and the results show that low bids lead to long deadlines for job execution. The analysis carried in [9] shows that check-pointing times below one hour can significantly decrease costs incurred due to job termination, although in some instance types check-pointing time can be extended above one hour. Tang et al. [10] propose a bidding strategy for Amazon spot prices that minimizes costs for given time constraints. Kushwaha et al. [11] perform an extensive simulation analysis of the spot market analysis on US East and South Asia regions. However they only consider bidding levels close to the on-demand price, do not consider adaptive bidding nor check-pointing of the computation state.

The Amazon EC2 spot instances are particularly suited for scientific computing, e.g. in [12] a genome sequencing is considered – the authors analyze dependencies between bid levels, number of virtual processors and sequencing time and costs. Another approach discussed in [13] is to utilize reserved instances for long-term running process and to use spot instances for short-term processes.

The existing approaches have several limitations. Firstly, some authors (for instance Javadi et al. [7]) perform statistical analysis only. However, due to the complex spot pricing mechanism this information is insufficient for decision making regarding bidding decisions.

Secondly, Amazon is constantly changing its pricing policy. For example, major recent changes have included around a 30% price reduction on April 1, 2014, introduction of 2 min spot instance termination notice on January 6, 2015 [14] and introduction of new C4 instances on January 11, 2015. The changes in the pricing scheme may lead to previous analyses no longer being valid. For instance, earlier results which claimed that bidding too low leads to a small increase in savings but huge increases in computational time (e.g. Andrzejak et al. [8] point out that *"(...) bidding low prices reduces the monetary cost typically only by about 10% but can lead to extremely high execution times (or, equivalently, realistic deadlines) – up to 400x the task length (...)"*), are no longer true under the current price patterns. Thirdly, some papers (cf. [10]) assume that the spot price changes hourly and a decision to perform calculations at a given price is made hourly as well. Moreover, the above papers do not consider the possibility of obtaining free time by bidding close to the current low price and having a good chance of an out-of-bid situation. Finally, the papers do not analyze how the need for check-pointing influences spot bidding strategies.

In this paper we focus only on an end-user perspective. Our aim is to prepare a realistic model of the spot pricing market with flexible characteristics for simulation experiment features. In this way, we remove all the limitations of models proposed in the literature. We propose a cloud simulation framework that emulates the Amazon EC2 spot billing mechanism exactly and takes into an account the following factors that significantly determine the costs of running large scale simulations:

(1) ability to bid for many different instances and switch between them;
(2) instance booting delay;
(3) real time reaction to instance termination by Amazon;
(4) exact modeling of the Amazon billing mechanism (in particular, obtaining computing power at no cost when calculations are terminated by Amazon);
(5) varying time between subsequent simulation state check-pointing events.

The proposed optimizer fulfills the above requirements and is designed in such a way that it can be applied to guide real-life bidding on Amazon EC2.

The remainder of the paper is organized as follows. After the Introduction in Section 2 we discuss the literature related to our research. Next, in Section 3, we shortly describe technical characteristics of how simulations in Amazon EC2 are performed. In Section 4, we provide the details of Amazon EC2 billing rules and outline the design of our cloud simulator which is implemented in Python (source code available at http://bogumilkaminski.pl/pub/cloudspotsim.zip). Next, in Section 5, we perform an initial analysis of cost-time trade-off in simulation execution. In Section 6, we discuss a mathematical formalism

that describes the adaptive model of computations on spot-price markets. Finally, in Section 7, we present the results of our simulation on real data taken from Amazon EC2 to show that our approach is valid.

2. Related work

The cloud computing paradigm (*Infrastructure as a Service*) is gaining increasing popularity due to highly competitive costs compared to employing on-site infrastructure. With Amazon currently being the main player in the market there exists a constantly growing body of literature on computing cost optimization. We can divide the literature on cloud computing infrastructure cost optimization into the following 5 areas:

(1) *technical* – handling of technical issues that become more apparent in IaaS scenarios (state check-pointing, network communication, extending a traditional HPC grid with cloud computing [5,6,13,15]);
(2) *benchmarking* – benchmarks and usability reports of IaaS virtual hardware performance [16,17], measuring instance startup times [2];
(3) *statistical* – analyses of the EC2 spot market dynamics with statistical tools, often accompanied by bidding recommendations [7,18,12,13,16];
(4) *market design* – attempts at reverse engineering the current market mechanisms or proposing new market designs [3,4,7,19];
(5) *pricing simulation* – simulating of spot market pricing [8,9,11].

The *technical* area focuses on searching for technically feasible and cost-efficient solutions that can actually make cloud computing work. The results presented in [15] focus on supporting a grid with the cloud and the authors do not attempt to optimize EC2 bidding strategy. In [13] an analysis of technical aspects of spot, on-demand and reserved instances is presented and utilization recommendations are made, however, it is not backed by computational analysis.

The *benchmarking* area focuses on measuring instance booting times, actual vCPU efficiency and network transfers. Those analyses are often done with real-world workloads. Iosup et al. [16] analyze computing speed variance while in [2] instance startup times are measured experimentally. El-Khamara et al. [17] provide a comparison of HPC workloads in MPI settings. The authors point out that a variance arises in communication times in MPI workloads. We suspect that this variance arises from the multiple tenancy mechanism used by Amazon [1]. In this paper we consider a model for single threaded running massively (eg. simulation optimization).

Statistical papers focus on the analyses of dependencies of spot prices and subsequently try to make recommendations on market bidding. Javadi et al. [18] analyze intercept times for spot instances. However, they do not propose any actual bidding strategy. This paper is further expanded by the authors [7] where they use a mixture of Gaussian distributions in order to model EC2 spot price patterns. Angiuoli et al. [12] analyze profitability of spot instances for genome sequencing.

Market design papers try to explain how the pricing mechanism works [3] or make market design postulates [4]. The paper [20] stresses the importance of the integration of cloud markets in order to increase their efficiency. The authors propose dynamic pricing for cloud instances and show that it is more efficient - both from a users and providers perspective. Yet another approach is presented in [19] – an EC2 market analysis is performed and the authors postulate that spot prices are not demand-driven but are instead randomly changed by Amazon in order to increase its profit. However, this claim does not appear to be supported by Amazon's efforts to reduce spot price volatility—e.g. the recent action of Amazon to send a two minute notice before spot instance shutdown in order to allow for graceful termination [14].

The *bidding simulation* area papers focus on building simulation models that allow for testing various spot bidding algorithms. In [11] only bids close to the on-demand price are considered and there is no analysis of computation state check-pointing. Another approach is to consider time constraints as an SLA and try to minimize costs – see [10,8]. Yi et al. [9] show that check-pointing strategy is an important factor in bidding decisions. They propose check-pointing strategies based on an hour-boundary and rising edge. *Hour boundary* check-pointing means saving the state at the end of each hour. *Rising edge* means observing price patterns and saving a check-point when the current price raises and is likely to be beyond the bid price. Next, they propose different adaptation mechanisms for those strategies. However, the authors focus only on running a task just on a single instance type – they do not consider adaptive switching between instance types. In the hourly boundary policy (in their model it turned out to be optimal for computational instances) they consider check-pointing only at the end of the hour. It should be noted that hourly check-pointing is not always optimal – they neglect the fact that the decision to save computation state should also depend on check-pointing costs. In our paper we show that, in many scenarios, cross-instance bidding and allowing non-hourly check-pointing is the optimal bidding strategy.

The presented literature survey shows that all earlier models of Amazon spot market had simplifying assumptions limiting the ability to compare different bidding strategies. Therefore, in this paper we propose a cloud simulation framework that emulates the Amazon EC2 spot billing mechanism exactly. This allows us to develop novel bidding strategies that have previously been considered in the literature.

It should be noted that we have constrained our analysis to renting virtual machines in the cloud (probably being run on Linux and self-configured by a person performing computations) rather than using an SaaS approach, although such software

for simulation exists, for example, the Model Exploration Service (MES) proposed in [21] that is offered by AITIA and runs on Amazon cloud infrastructure.

3. Technical characteristic of simulations in the cloud

The goal of this section is to provide technical assumptions for running large-scale simulations in the cloud. These assumptions lay down the foundations for constructing a decision problem for the optimal spot-price bidding strategy.

Amazon offers public cloud computing services in 9 regions (3 in Asia, 2 in the European Union, 1 in South America and 3 in the United States). Each region contains two or three availability zones that represent physical server locations. At each availability zone several virtual machine instance types are offered. The machine types differ by their purpose and can be classified as: general (with fixed and variable computing power), compute-optimized, memory-optimized, storage-optimized and GPU mode. For the analyses of cloud simulations we are going to focus on compute-optimized and fixed computing power general instances—we do not consider variable computing power instances since they offer only a small hourly computing budget.

A spot instance is started with a given bid level on a particular machine type in an availability zone. While the instance is running, the spot price for the machine is continuously changing due to changes in supply and demand for the computing power. Whenever the spot price exceeds the bid price a spot instance is *terminated* (shut down) by Amazon. If a spot price is equal to the bid price, Amazon reserves the right to decide whether to terminate an instance or not. However, spot prices are often several times lower than on-demand prices – and this makes them attractive for applications in simulation modeling.

Liu [22] points out that for some applications the abrupt process termination is not acceptable. However, for experimenting with simulations one can either run many short jobs or else introduce a check-pointing mechanism for longer simulations. In both cases the problem of computation termination is not particularly onerous. In simulation modeling, the following check-pointing mechanisms can be applied (also see [1]): Elastic Block Storage (EBS), Simple Storage Service (S3), Relational Database Service (RDS).

In all of the above scenarios, the cost of storage is relatively low in comparison to computing power costs and can be ignored in large scale simulation settings. Hence, for most applications, EBS seems to be the most attractive option, while S3 might be required where simulation state snapshots are expected to re-initiated in different regions.

It is not always valuable to use cloud services to run simulations as it adds complexity to the execution work flow. Most of the time we would want to utilize such an architecture when we have a large volume of computations to be performed. To be more precise, in simulation modeling we consider two simulation problem types that generate high computation time requirements:

(1) *small simulations* – the time needed to finish computations in a simulation is short (e.g. 1 min on a single core, so there is no need for check-pointing) but we need to execute the simulation repetitively for a very large number of design points and/or with a large number of repetitions for each design point (in the order of millions and more);
(2) *large simulations* – a single simulation runs for several days (possibly on many cores) and has to be check-pointed to safeguard against instance termination; the number of required simulation repetitions can vary (usually it is lower than the number for small simulations).

In this paper we will analyze both scenarios through the appropriate setting of simulation parameters. Let us now proceed to a description of the Amazon spot market.

4. Simulating the spot instance price mechanism

The goal of this section is to describe in detail the spot pricing mechanism and explain the design of a simulator that allows us to test bidding strategies.

4.1. Spot instance cost charging mechanism

The Amazon EC2 spot instance billing mechanism is complex due to everchanging prices and the possibility of instance termination – either by the instance user or Amazon.

The process starts with a user bidding a price for a spot instance for a selected machine type in a given availability zone. The instance is created only if the bid price exceeds the current spot price. When the bid price is equal to the current spot price Amazon reserves the right to either start or not to start an instance. We do not consider this scenario here as it has no significant influence on our analysis.

At startup, the billing price is determined as a spot price at instance creation time. Next, the billing algorithm works in hourly granularity while the spot prices are changing constantly in response to changes in supply and demand for computing power. Hence, the computing hour can be interrupted – either by a user or by Amazon.

If an instance is interrupted in the middle of an hour by a user, she is still obliged to pay for a full hour. Amazon interrupts an instance where the instance type spot price increases and exceeds the bid price. In the case of such an interruption the user does not pay for the partial hour.

If a whole hour passes without any interruption then the user has to pay for this hour and a new billing price is determined as the spot price at the start of the new billing hour.

It is important to note that the user is not allowed to change the bid price after the instance is created.

From the above description we can draw two conclusions. Firstly, users can benefit when Amazon terminates their instance as they get some computation time free of charge. On the other hand, every termination of an instance by Amazon means that a new instance will have to be started in the future and each time an instance is initiated there is a booting time that does not count for simulation execution.

4.2. Cloud simulator design

In order to analyze the different possible bidding strategies we have created a tool – the EC2 cloud pricing simulator. The simulator was created with the Python programming language. The tool consists of a single class that operates on historical data and emulates the Amazon EC2 pricing mechanism.

The main method provided with the simulator is `estimate_cost_s`. This method calculates total costs for a bid at a specific zone and machine. The bid starts at a given time *sta_s* and the machine will be unconditionally terminated at time *end_s*. The *boot_time_s* represents seconds taken to boot the machine and initiate the simulation. The user can request either a concrete server time or a number of simulations. If the requested calculation ends the machine can either immediately terminate or continue working to the end of the final hour. The parameter *stop_on_terminate* indicates whether after instance termination the simulation should stop or whether the instance should be launched again as soon as the spot price again drops below the bid price.

The work performed by the EC2 price simulator starts with loading a file containing historical pricing data. In the source code we provide a tool that generates such a file utilizing the Amazon `boto` library. Once the price data is loaded into memory, a user can utilize our EC2 library methods to calculate costs for various scenarios at different zones and for different machine types.

The implementation of the method `estimate_cost_s` follows the Amazon billing algorithm [1]. It is a discrete-event simulator where subsequent events are triggered by spot price changes.

The basic implementation of the billing algorithm has been further utilized to test various bidding scenarios. In this paper we consider two types of bidding strategies: (1) static fixed-bid and (2) adaptive bidding.

The analysis of the prices for the *fixed-bid* scenario for a given zone and machine type can be simply accomplished by a single call to the `estimate_cost_s` method. This function for a given, region, instance type, bid level and time range calculates costs as well as machine availability time that be obtained by bidding at a specific level. Hence, the procedure for calculating availability time and computing cost takes into consideration the situation in which the acceptable bidding level is too low and a user waits for the spot price to drop in order to place a bid that will result in starting an instance.

The *adaptive scenario* dynamically changes bidding depending on two market conditions: spot price level and price volatility. We use our simulation tool to test algorithms where we bid marginally higher above the spot price with a reservation level (i.e. do not bid if the spot price is too high). For this analysis we set the value of *stop_on_terminate* to `True`. After a machine is terminated a new machine needs to be started at a new bid level. We consider an adaptive bidding scenario with switching across different zones and machine types. It should be noted that switching zones might be not cost-efficient for computing scenarios that require huge amounts of data to be transfered (the current EC2 data transfer cost is $0.01/GB for transfers within a single region and $0.02/GB for cross-region transfer).

The static scenario was used to carry out simulations in Section 5 while the dynamic scenario is analyzed in Section 7.

Let us now move to the analysis of spot prices patterns on Amazon EC2.

5. Characteristics of spot price dynamics

In this section we present statistics for pricing level changes on Amazon EC2 and provide an initial analysis of possible bidding strategies.

5.1. Price patterns on the spot market

In this subsection we discuss results of our investigation into Amazon EC2 spot prices. For the purpose of this analysis the price for a given availability zone and machine type is called an *offer*. We analyzed prices from 2014-04-02 to 2014-07-06. In this period there were no Amazon pricing policy changes. The recent significant price reductions was announced by Amazon on April 1, 2014 [23].

Virtual servers offered by Amazon differ greatly in the computational power they offer. The power of Amazon machine types is measured in EC2 Compute Units (ECU). The ECU used to have the following definition: *"one EC2 Compute Unit provides the equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor"* (the original Amazon statement is no

longer available on Amazon web site; however it is still available at other sources including Wikipedia). In order to ensure comparability of prices for various machine types we standardize machine spot prices by dividing them by the ECU value. This transformation means that we assume that there is no issue of performance loss due to parallelization of computations. This assumption is true for simulation models where single-thread simulations are being run several times (e.g. applying Monte-Carlo methods for a stochastic model or sweeping a model parameter space).

It should be noted that the ECU is not an exact measure of computing power. A large variance in MPI computation times for HPC workloads can be observed experimentally [17]. This issue is likely caused by multi-tenancy – a single virtual computer can be run on more than once hardware instance. Schad et al. [24] point out that Amazon cloud performance shows a 21–24% coefficient of variance and, moreover, the performance clearly falls into two clusters – which could possibly be a due to using different hardware to host computing instances. In a more recent paper [25], the authors discover that a virtual EC2 instance can be backed up by one of 6 types of processors, resulting in markedly different performance. In our approach, we make two assumptions regarding the observed ECU variance. Firstly, we limit our analysis to simulation models that can be run in a single thread without synchronization between threads – i.e. we consider Many-Task Computing (MTC) [16,26], which is typical in many simulation applications including parameter sweep, and hence we do not consider MPI models. Secondly, we assume that the ECU measure represents an expected value of a node performance. Since an HPC cluster is constructed of several nodes, our goal is to optimize the average expected cost of running computations.

It is natural to start the analysis of market prices with consideration of their mean and volatility. In Fig. 1 we present mean spot prices for all considered offers. We observe that most of the offers have relatively similar averages, but there are exceptions. For example the us-east-1c zone is systematically expensive, m3-type instances are expensive and the most attractive offers are for c3.large, c3.xlarge and c3.2xlarge instances in us-east-1a, us-east-1b, us-west-2b and us-west-2c availability zones. Additionally, we have calculated that average prices are highly correlated with standard deviations of prices (correlation equal to 82.5%). However, they are not correlated (correlation equal to 0.86%) to their coefficient of variation (standard deviation divided by the mean). This implies that the price volatility scales approximately linearly with mean prices. Additionally, we note that the observed spot price range is very large – from 0.002286 USD/(hour×ECU) to 0.5384 USD/(hour×ECU).

The analysis of mean prices would constitute good guidance for a computation time allocation if we considered static allocation of computing power. However, we are interested in a dynamic algorithm. Such an algorithm allows for active switching between instances, thus we are more interested in the probability that a given offer has the lowest price across the entire market than the mean price of the offer, as this is the offer we will want to choose. Similarly, instead of the standard deviation of prices it is better to analyze the probability of getting a "free lunch" from Amazon. By this term we mean a situation when we run a server for at least 5 min (server booting time) and Amazon terminates it before one hour passes. It should be noted that making use of the free lunch mechanism may require renting additional persistent EBS storage space that is not free. However, the cost of 1 GB of EBS is 0.1$ per month or 0.00014$ per hour. This cost can be further reduced to 0.03$ per GB per month when using S3 for persistence (data transfer to S3 within the same zone is free). Hence, in our model we assume that the additional storage space demand that arises from check-pointing is relatively insignificant and can be neglected.

In Fig. 2 we analyze the probability that a given offer has the lowest spot price. In fact, in computations we regard an offer as optimal if it costs no more than 0.00001 USD/(hour×ECU) above the lowest spot price, since such a minuscule price difference can be considered insignificant (it is equal to 10% of the granularity of Amazon prices which are given with four decimal places). This means that the probabilities across all offers sum up to more than 1. Given this transformation of the

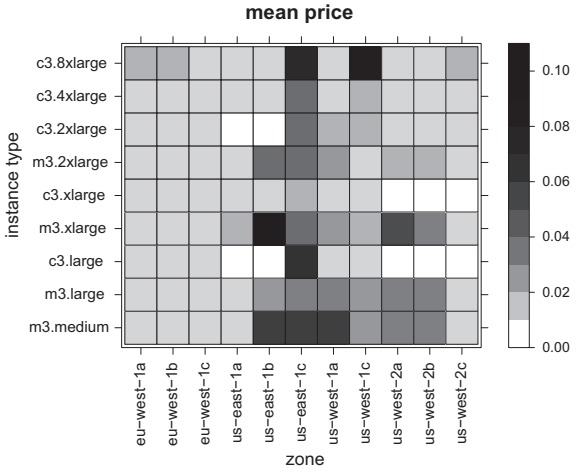

Fig. 1. Comparison of average spot prices in USD/(hour×ECU) by offer. Spot prices are weighted by time of their applicability.

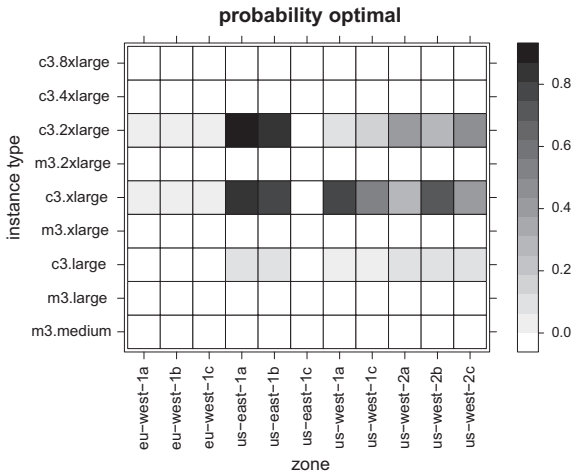

Fig. 2. Percentage of time that a given offer has the optimal spot price. At any given moment in time, the spot price of an offer is called optimal if it costs no more than 0.00001 USD above the lowest spot price.

approach we have a much more precise view of the market. We can conclude that one should concentrate on bidding only for c3.xlarge and c3.2xlarge on us-east-1a and us-east-1b zones when their prices are low.

A fresh look at volatility is presented in Fig. 3 where we analyze "free lunch" probability. We used the historical data to calculate how likely a bid at the spot price level (plus small value in order to ensure that Amazon actually initiates the instance) is going to end up with instance termination during the first hour. More precisely, for a given bid level we calculate a percentage of computations that were terminated within 60 min due to price increase beyond the bid level. We can observe that there are some relatively stable offers – such as the m3.2xlarge instance in the eu-west-1a zone, where there is almost no opportunity for termination of computations by Amazon. On the other hand, there are servers such as c3.4xlarge in the us-east-1b zone, where the free lunch probability reaches approximately 40%. On such a server we could try to exploit the characteristics of the Amazon billing model in order to minimize computing cost. Such an analysis is performed in Section 7.

5.2. Cost-time tradeoff for computing simulation

Before we move on to the description of an adaptive bidding algorithm let us start with a simpler scenario.

We want to verify whether there exists a cost-time trade-off for computing simulations on Amazon EC2. In order to do this we perform the following experiment. Assume that there is a single decision variable – a price that will be bid by the user for the entire period and it is a fixed price (there is no adaptive bidding). We define a reservation price as the multiple of the minimum price observed on the market. We call the multiple value the reserve-price-multiplier (*RP multiplier*). In our

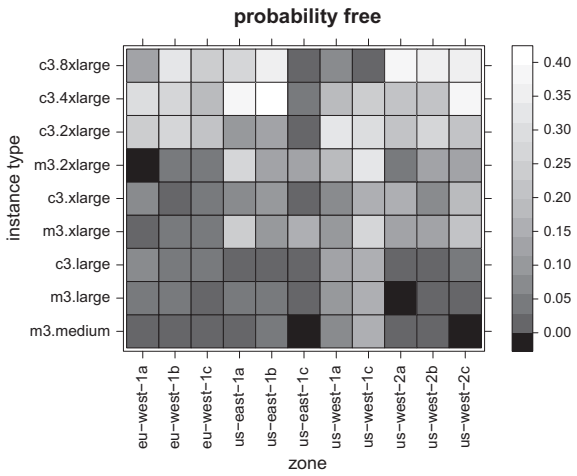

Fig. 3. Comparison of expected probability of free lunch.

analysis the base for the multiplier – the minimal observed price is 0.002285714 USD/(hour×ECU), which translates to 0.05485714 USD/day. We set the RP multiplier to range from 1 to 10.9 with 0.1 step. We always bid on a single offer (single machine type in a single availability zone) where the reservation price is higher than the spot price. The bid level is just above the current spot price.

We perform the experiment by running a simulation for 61 days (from 2014-04-10 to 2014-06-10) and measuring:

(1) expected cost in USD per 1 ECU of 1 day of effective simulation time (if we were able to bid the minimum observed price for the whole period of 61 days then the minimum price per day would be equal to 0.05486);
(2) expected days of effective simulation time (it will be 0 if we bid below the historical minimum, otherwise up to 61 when the bid price is above the historical maximum).

All subsequent calculations refer to cost per 1 ECU and so this is omitted unless clarification is necessary.

In order to compute the effective simulation time, we subtract the booting time (5 min) every time a new instance is initiated. Hence, the simulation time is the time actually available for computations. As this is a simplified analysis we disregard problems with check-pointing of the simulation state (which is also a time consuming task that can be considered a loss from a simulation execution perspective).

Given the above definitions the decision maker faces a bi-objective optimization problem: minimization of expected cost and maximization of expected days of effective simulation. In Fig. 4 we present a scatter plot of USD/day and total time of all offers for all considered RP multipliers. The gray curve denotes the Pareto frontier. It is evident that there is a vast space of highly inefficient combinations of offers and RP multipliers. Additionally, it should be stressed that the Pareto frontier is very steep.

Let us consider *extended dominance for bidding strategies*: a bidding strategy is defined as not dominated subject to extended dominance if it is not dominated by a combination of other bidding strategies used in some proportion. For example, assume that we have only three alternative pairs of cost per day and total time $(1,10),(5,11)$ and $(7,14)$. They are all Pareto efficient. However, if we use the strategy $(1,10)$ for 50% of the time and the strategy $(5,14)$ for the remaining time our total computation time will be $(10+14)/2=12$ which is more than 11 but the average cost per day is $(1\times 10/2+7\times 14/2)/12=4.5$ which is less than 5. This means that $(5,11)$ is dominated subject to the extended dominance.

It should be noted that for our simulation results, it can be determined that if we allow only points that are not dominated subject to extended dominance than there are only three Pareto efficient combinations (USD/day, total time): $(0.04670,1.2847),(0.05497,60.9389)$ and $(0.05501,60.9965)$. This implies that a roughly 1.177 times increase of the cost produces a 47.43 times computation speedup. This means that, unless we are very price sensitive, we can obtain almost 100% of time efficiency at a reasonably low price if we properly select a combination of offer and reservation price.

Let us also note that the cheapest server type in the Pareto frontier offers costs of 9.77% below the minimum (costs are lower due to free lunches) and with time constituting only 31.06% of the maximum possible time. On the other hand the most expensive instance offers 4.45% higher costs than the minimal costs with effectively full utilization of 61 days for running the instance.

Let us now investigate the effects of offer and reservation price changes on cost-time trade-off.

Fig. 5 presents an interaction of RP multiplier(i.e. bid level) and mean incurred daily costs (in USD per ECU) and machine availability time within two months. We can see that in general – increasing the RP multiplier increases cost and improves total time. However, most of the time benefits are reaped by relatively low increases of the RP multiplier. A further increase steadily drives up USD/day but total time becomes stagnant.

The analysis by offer (combination of zone and instance type) shows a large variability in mean USD/day and mean total time (averaged over RP multiplier). The mean USD/day ranges from 0.0550 to 0.2828 but most of the values are clustered

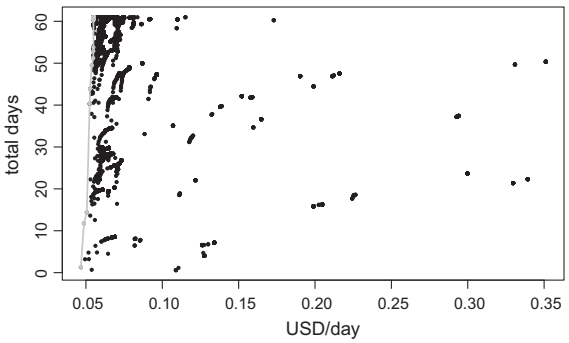

Fig. 4. Scatter plot of USD/day and total time of all offers for all considered RP multipliers. Gray curve denotes the Pareto frontier.

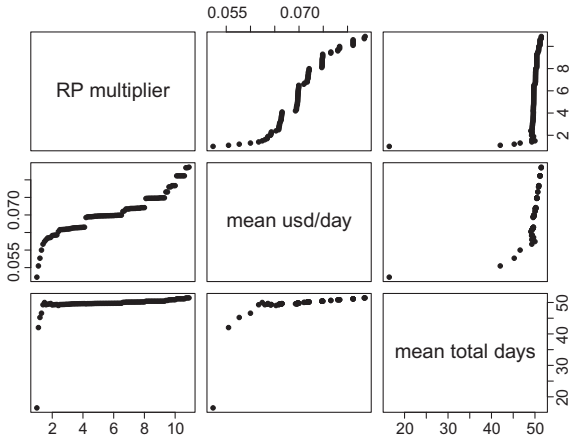

Fig. 5. Dependencies between the level of RP multiplier (i.e. bid level), mean incurred daily costs (in USD per ECU) and machine availability time within two months. The simulation results are averaged over offer.

around the lower bound (the median is 0.0644). Similarly, the mean total time ranges from 8.294 to 60.997 but its median is 59.338 (we do not provide a visualization here as there is no natural ordering of offers).

In order to verify which factor has the biggest influence on 'USD/day' and 'total time', we have built two random forest models [27] predicting those variables. The obtained random forests performed satisfactorily, explaining 81.69% and 94.32% of target variable variance respectively. In Fig. 6 we show variable importance plots for both models. We can observe that the zone is the most important factor and the RP multiplier is relatively unimportant. The machine lies somewhere in between – being slightly more crucial for explaining the total time.

This implies that the most important decision in bidding strategy is the choice of an appropriate offer (and especially zone choice) and proper setting of the reservation price has a secondary (though not negligible) influence on the results. We will confirm these initial observations in Section 7, where we analyze the performance of the dynamic bidding algorithm presented in Section 6.

6. On-line optimizer of computation cost model

In the adaptive bidding scenario we want to take the following observations made in Section 5 into account:

(1) there is a high variability of prices between zones and between instance types;
(2) bidding high above an observed spot price leads to a large cost increase without a significant decrease in a computation time.

Based on these characteristics we consider a setting where different offers (server types and/or locations) are taken into account simultaneously, but at any given point in time the cheapest-per-ECU offer is chosen to start computations. However,

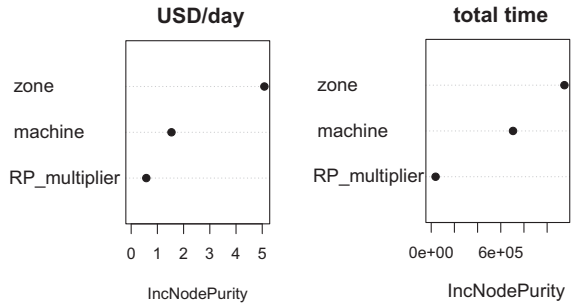

Fig. 6. Variable importance in random forest [27] models explaining USD/day and total time. The higher IncNodePurity value for the variable the more important it is in the model. The IncNodePurity measure is defined as decrease of residual sum of squares obtained by splitting on a given variable averaged over all trees in the random forest, see [28] for a detailed discussion. We use randomForest package [29] from GNU R [30] to compute it.

when the spot price is above the reservation price no bid is made (the algorithm "waits" while the spot price is too high for all considered offers).

We additionally assume that the available computing power is fixed throughout the simulation. This follows the constraints that Amazon sets on the number of running machines. Otherwise, we would wait till the price is low and start thousands of processes – which is what Amazon wishes to avoid in order to reduce spot price fluctuations. In fact, even such an aggressive strategy is not necessarily optimal, since we would pay a penalty for server booting times for thousands of servers. The default cap is five instances per region, but the limit can be increased by filing a request to Amazon. In our scenario we assume that we request 32 virtual cores (vCPUs) available for simulations.

In this analysis, similar to the approach presented and discussed in subSection 5.1, we standardize the prices by dividing the machine price level by its the ECU power. Hence, all costs are presented for one ECU-hour.

The simulation request by the user can be characterized by the following parameters:

(1) computing time required to finish one full simulation;
(2) number of required simulation runs;
(3) check-pointing time (time to store a simulation inner-state; this is important only for long simulations; in short simulations we assume no check-pointing).

Given the assumptions presented above and the results from the analyses of spot price dynamics, the user of an adaptive billing algorithm has to make the following decisions:

(1) maximal spot price she is willing to accept per 1 ECU-hour denoted as $\overline{B}$;
(2) $\mathcal{B}_s = (\mathcal{B}(1), \mathcal{B}(2), \ldots)$, a sequence of user bids; a bid $\mathcal{B}(i)$ is a tuple (t, n, b) consisting of three elements: bid time t, offer number n and bid level b (remember that by an offer we understand a combination of server type and location; for simplicity of notation we assume that we assign a number to each Amazon offer);
(3) F, the frequency of check-pointing (how often do we want to perform check-points counting from the start of the simulation).

Notice that the bidding sequence $\mathcal{B}_s$ will be dynamic in general. We assume that if the bid $\mathcal{B}(i)$ is successful (a new instance has been started), the next bid $\mathcal{B}(i+1)$ is not placed until the previous instance is terminated by Amazon. For example in Section 5 all bids $\mathcal{B} \in \mathcal{B}_s$ had fixed offer n and bid price b but bid times t were determined dynamically.

The user has the following two objectives: (1) minimize the total computation cost and (2) minimize the total computation time.

In our optimization process we consider, in particular the following facts:

(1) if Amazon terminates the simulation that has been check-pointed we get a free lunch (that portion of the last hour of simulations until the last check-point);
(2) after a bid is established we continue the computations on the same server as long as either the simulations are completed or the server is terminated by Amazon;
(3) computation time starts after the time required for booting
(4) computations are suspended during state check-pointing (and hence overly frequent checkpoints reduce available computation time).

The adaptive bidding mechanism is presented in Algorithm 1. Two parameters drive the bidding decisions – spot prices and frequency of price changes. At each point in time a group of offers is considered and the one currently cheapest is chosen. If the spot price of the chosen zone and machine is lower than the reservation price a bid is made. Next, on the basis of the historical data an average distance between price changes is calculated over the last 6 h. This distance is subsequently used to set the space between price changes. In this way we ensure that we check-point more frequently on servers with highly volatile prices and less frequently on more stable servers.

The adaptive bidding procedure is called each time an instance is terminated by Amazon. If it is possible (bid $\mathcal{B}$ was returned), a new instance is created. Otherwise, the algorithm waits until the spot price on any of the considered offers drops below $\overline{B}$ and then calls the adaptive bidding procedure again.

Algorithm 1. Adaptive mechanism for bidding for long simulations at given time t (for short simulations it is idencical but we assume no check-pointing and no loss of data due to instance termination). By $spot\_price(t, offer)$ is spot at considered time for given *offer*. Long term minimum price *ltm* is equal to 0.002286 USD/(hour×ECU). ε is a small value that is added to spot price in order to ensure that the instance will be created by Amazon. Function $price\_change\_freq(t, offer)$ calculates how often prices changed for *offer* during six hours before time t.

```
Procedure ADAPTIVE_BID(t, rp_mul, cp_mul, offer_list)
    best_offer ← min_{offer∈offer_list} spot_price(t, offer)
    B̄ ← ltm × rp_mul
    if B̄ ⩾ spot_price(t, best_offer) then
        B ← (t, best_offer, spot_price(t, best_offer) + ε)
        F ← price_change_freq(t, best_offer) × cp_mul
        return bid B with check-point frequency F
    else
        return no bid
    end if
endprocedure
```

We can see that the crucial meta-decision to be made is on the billing instance group (list of offers). We consider four types of groups:

(1) single instance;
(2) all instances in one zone;
(3) all zones for one instance type;
(4) all offers (all zones and all instance types).

The rationale behind choosing instances in one zone is that the data transfer within a single zone is very fast. Moreover, we want to check how the price correlation within zone influences the cost and time of simulation runs. The analysis of one instance type across zones can explain whether it is worth running simulations on multiple regions and zones rather than just concentrating on a single zone. Additionally, it is reasonable to assume that the simulation can be fine-tuned to work on certain machine types. In such a case, using it across all zones is reasonable.

In Section 7, we show that the above scenarios allow us to choose very competitive options – which are both cheap and fast.

7. On-line optimizer simulation results

The analysis of the spot price mechanism in Section 3 leads us to the conclusion that two distinct simulation scenarios should be analyzed: short-time simulations being run a large number of times and long-time simulations that can take several days in order to finish a single simulation run.

Subsequently, in the following two subsections, we discuss how different reservation price levels and check-pointing times influence the time and costs of both simulation types. Finally, in the third subsection we perform sensitivity analysis and show that our approach is valid for different time periods.

7.1. Short simulations

We begin our analysis with the short-run simulations. Based on the results from SubSection 5.2, we know that it is efficient to bid near the current spot price. Recall that in Algorithm 1 we define the bid reservation price $\overline{B}$ as a product of the RP multiplier and the minimum observed price, 0.002285714 USD/(hour×ECU), but this is not a bid price. This analysis is different to Section 5, where reservation price was equal to bid price. We analyze single offer scenarios ($9 \times 11 = 99$ scenarios), 9 scenarios for server types, 11 scenarios for zones and one scenario where all offers are considered. The reason for such a choice of scenarios is that we know from SubSection 5.2 that the results are highly sensitive to the selection of zones and machines.

In the short simulation scenario we consider massive running of short one-minute simulations. For simplicity, we assume that the one minute simulation includes storing the outcome – the outcomes are stored immediately after the simulation ends. Moreover, we assume that storage costs can be neglected. We simulate our algorithm for 61 days starting from April 10, 2014 and measure two outcomes: simulation time and daily costs per 1 ECU of computing power. When considering the long-time minimum price per ECU-hour of 0.002286 USD, we can calculate the minimal cost as 0.05485 USD (this assumes no termination of computations and thus no free lunch received). Please note that the minimal cost is 15.238% of the on-demand cost of running c3 type instances (the on-demand hourly per-ecu cost for c3.large, c3.xlarge, c3.2xlarge is 0.015 USD, for c3.4xlarge: 0.015273 USD and for c3.8xlarge: 0.015556 USD). The maximum simulation time that can be obtained is 60.99653 days (i.e. 61 days minus 5 min required for instance startup).

Fig. 7 presents a scatter plot of USD/day and total time of all offers for all considered RP multipliers. The gray curve denotes the Pareto frontier. It is similar to the results for single instances presented previously in Fig. 4. The Pareto frontier

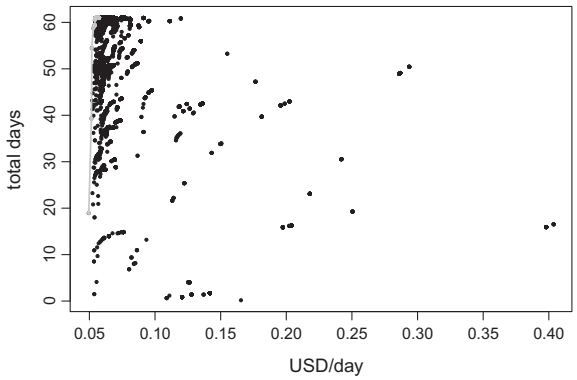

Fig. 7. Scatter plot of USD/day and total time of all offers for all considered RP multipliers. Gray curve denotes Pareto frontier.

is presented in detail in Table 1 (note that we give rp_mul ranges, since for varying values of this parameter identical values were obtained).

For the bidding strategies included in the Pareto frontier we have verified that the first two low-cost short-time single server scenarios give very unstable results and we can see that the high-cost long-time single server scenarios do not add much in actuality. Thus, in practice it is only worth considering offer combinations consisting of c3 servers of one type grouped over all zones or all servers.

Because of this observation we present a simulation of costs for utilizing the cheapest c3 type server from all available zones (see Table 2). We calculated the costs for each server type and additionally we considered a scenario where bidding is made across all available servers. The analysis was carried out for various spot price reservation levels (rp_mul). However we assumed that the reservation price is lower than the on-demand price. The analysis of results in Table 2 leads to the following conclusions:

(1) utilizing a group of c3 servers across zones gives stable results – we can almost always find a server with a very low price (less than 1.1 rp_mul);
(2) rp_mul is not critically important – increasing it only slightly increases the price and computation time. If rp_mul is more than 1.2 the results do not change;
(3) increasing server power reduces cost marginally but also slightly reduces time; The differences are not crucial $\approx 6\%$ for cost and $\approx 12\%$ for time.

In summary, we have found robust scenarios that are time and cost efficient. The simulated values are very close to the best theoretically achievable values (total time equal to 61 days and USD cost per day equal to 0.05485714). Additionally, the scenarios are flexible – one can decide on server type without practically any loss and tune the simulation to the available number of cores.

The scenarios differ only by single percentage points from the theoretical optimum (61 days for 0.05485 per ECU-day). Hence, the proposed approach of bidding just above the reservation price can be utilized for actual simulation run scenarios.

Table 1
Pareto frontier for short simulation.

Scenario	USD/day	Total time	Min(rp_mul)	Max(rp_mul)
us-west-1a_m3.2xlarge	0.0495	18.9444	1.1	1.1
us-west-1c_c3.4xlarge	0.0516	39.2993	1.1	1.1
c3.8xlarge	0.0519	54.4924	1.1	1.1
c3.8xlarge	0.053	58.75	1.2	1.2
c3.8xlarge	0.0541	59.3632	1.3	10.2
c3.4xlarge	0.0542	60.4951	1.2	1.2
c3.2xlarge	0.0547	60.875	1.1	10.2
all	0.0548	60.9306	1.1	10.2
c3.xlarge	0.0549	60.9542	1.1	10.2
us-east-1a_c3.2xlarge	0.055	60.9847	1.3	10.2
c3.large	0.0552	60.9854	1.1	10.2
us-east-1b_c3.large	0.0553	60.9924	2.0	10.2
us-west-2a_c3.4xlarge	0.0573	60.9965	2.1	10.2

Table 2
Comparison of cost and time of short simulations being run on c3-type servers. In the columns, rp_mul scenarios are given. Scenarios greater than 1.2 are pooled as they were identical.

Scenario	1	1.1	1.2	>1.2
USD/day				
c3.large	–	0.0552	0.0552	0.0552
c3.xlarge	–	0.0549	0.0549	0.0549
c3.2xlarge	0.0544	0.0547	0.0547	0.0547
c3.4xlarge	–	0.0542	0.0542	0.0547
c3.8xlarge	–	0.0519	0.053	0.0541
all	0.0544	0.0548	0.0548	0.0548
Total time				
c3.large	–	60.99	60.99	60.99
c3.xlarge	–	60.95	60.95	60.95
c3.2xlarge	59.81	60.88	60.88	60.88
c3.4xlarge	–	60.45	60.5	60.53
c3.8xlarge	–	54.49	58.75	59.36
all	59.81	60.93	60.93	60.93

Table 3
Mean length in days of one continuous block of simulation per server type and rp_mul.

Scenario	1	1.1	1.2	>1.2
c3.large	–	15.25	15.25	15.25
c3.xlarge	–	5.08	5.08	5.08
c3.2xlarge	1.71	1.84	1.84	1.84
c3.4xlarge	–	0.41	0.43	0.45
c3.8xlarge	–	0.08	0.1	0.13
all	1.71	5.08	5.08	5.08

7.2. Long simulations

In the long-lasting simulation scenario we make an assumption that it takes 5 min to check-point a simulation and that the simulation restore process can take place within the server booting time. This is a realistic assumption, but can be also easily modified. In order to analyze the influence of check-pointing on scenario optimality, it is sufficient to note that we can consider distribution of uninterrupted blocks of simulations to determine how much time we would lose for simulation backups and how much computation would be lost because it was not check-pointed.

In this section, we concentrate only on those scenarios identified as efficient for short-simulations: that is, we bid for a selected server type across all zones. Under this assumption, we have calculated the mean time of continuous simulation blocks and present this in Table 3 (for rp_mul equal to 1, some servers are never started because their prices newer go low enough. Hence, there are missing values in the table).

We can observe that the servers differ significantly in continuous run-length. For example, c3.large is very stable and c3.8xlarge is very volatile. These differences lead to heterogeneity of the optimal time between check-points. It is calculated to an accuracy of 15 min and presented in Table 4. By the optimal time between check-points we mean the time that minimizes the total cost (for a particular bid level and instance type group) of handling possible instance termination – the instance time used to perform checkpoints and to reboot an instance that was terminated due to spot price fluctuations.

We can see that the stable servers c3.large, c3.xlarge and all (for rp_mul > 1) require much less snapshotting than volatile servers. This also means that with those machine types we will lose considerably less time in comparison to the no-snapshotting scenario considered in SubSection 7.1. The percentage of time lost is given in Table 5.

We can observe that on stable servers we lose much less time than on volatile servers. When we compare the percentage of time lost to the data from Table 2 we can see that this loss reduces effective simulation time and simultaneously increases the USD/day cost of the effective simulation time. Hence, it can be calculated that servers other, c3.large, c3.xlarge and all become ineffective.

The conclusion is that running short simulations is cost efficient on volatile, large servers. On the other hand, long simulations should be run on stable, smaller machines – otherwise too frequent check-pointing will become a significant cost that cannot be reduced by hunting for free lunches since the cost of an abrupt simulation termination is too large.

7.3. Sensitivity analysis

The goal of this section is twofold: (1) to analyze the stability of Amazon EC2 spot prices and (2) to validate the time-stability of the proposed bidding algorithm. We consider two data sets:

Table 4
Optimal time in minutes between check-points per server type and rp_mul.

Scenario	1	1.1	1.2	>1.2
c3.large	–	195	225	195
c3.xlarge	–	210	210	210
c3.2xlarge	195	165	165	165
c3.4xlarge	–	105	135	135
c3.8xlarge	–	90	75	90
all	195	210	210	210

Table 5
Percentage of available time lost due to check-pointing and simulation termination.

Scenario	1 (%)	1.1 (%)	1.2 (%)	>1.2 (%)
c3.large	–	2.86	2.55	2.86
c3.xlarge	–	3.08	3.08	3.08
c3.2xlarge	5.37	5.09	5.09	5.09
c3.4xlarge	–	10.17	10.61	10.36
c3.8xlarge	–	17.89	16.43	14.49
all	5.37	3.54	3.54	3.54

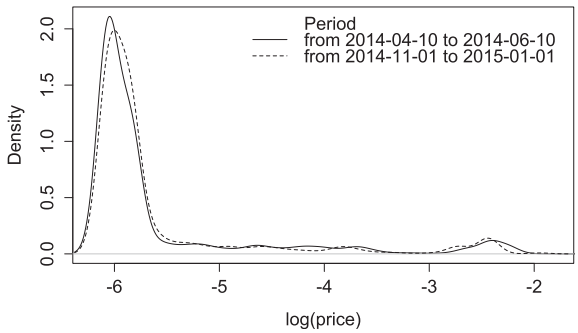

Fig. 8. The graph compares the density of average daily spot prices for the base and sensitivity period. It can be seen that the price distributions are very similar.

(1) *base period* – used to calibrate the algorithm (the data used in Sections 7.1 and 7.2). The data starts at April 10, 2014 and ends at June 10, 2014. This period was used to design and calibrate the bidding algorithm.
(2) *sensitivity period* – that data that we will use to validate the algorithm. The data starts at November 1, 2014 and ends at January 1, 2015. Hence, the sensitivity period covers Black Friday and Christmas, where the demand for computing power is larger than at any other time of year.

We start with comparing spot price distribution for the above two periods. Fig. 8 presents the price density for both periods. It can be seen that the distributions are almost identical. In the sensitivity period (that includes the peaks of Black Friday and Christmas) the prices are distributed slightly to the right. However, we have checked that throughout the whole period in some zones the bottom price of 0.002286 USD/(hour×ECU) is always available. Further comparison of standard statistics of base vs sensitivity period shows that they are almost identical: first quartile is 0.002301 vs 0.002334, median is 0.002547 vs 0.002789, third quartile is 0.002968 vs 0.002970 and mean is 0.008732 vs 0.008209 (notice that the mean is much higher than the third quantile as both distributions are highly skewed). Hence, we can expect that adaptive bidding results for those two periods will be very similar.

In order to verify the stability of our algorithm we simulated the adaptive bidding against the sensitivity period. We have checked that the Pareto frontier in the sensitivity period is almost identical to the frontier presented in Fig. 7. Additionally, we have calculated how the results presented in Table 2 for the base period change during the sensitivity period. We calculated the percentage differences of simulation costs and time between the compared periods and presented them in Table 6. We can see that the differences for almost all instances are very small. The biggest differences are for rp_mul equal to 1 for total time. However, this is to be expected since this means that we are bidding on the edge of termination of the computations so the outcome may be more unstable. Similarly, as the c3.8xlarge instance is the most volatile (which can be seen from data in Tables 3–5) the same effect can be observed there, but on a smaller scale.

Table 6
Comparison of percentage deviation of cost and time of short simulations being run on c3-type servers in the base and sensitivity periods. In columns are rp_mul scenarios given. Scenarios greater than 1.2 are pooled as they were identical.

Scenario	1 (%)	1.1 (%)	1.2 (%)	>1.2 (%)
% deviation of USD/day				
c3.large	–	0.12	0.12	0.12
c3.xlarge	–	0.33	0.33	0.33
c3.2xlarge	−0.48	−1.41	−1.41	−1.41
c3.4xlarge	–	2.58	2.64	1.77
c3.8xlarge	–	7.26	4.33	3.02
all	0.96	0.37	0.37	0.37
% deviation of total time				
c3.large	–	0.02	0.02	0.02
c3.xlarge	–	0.07	0.07	0.07
c3.2xlarge	−13.39	−0.27	−0.27	−0.27
c3.4xlarge	–	0.78	0.71	0.66
c3.8xlarge	–	7.98	1.48	1.48
all	−8.64	0.07	0.07	0.07

The above considerations show that the proposed algorithm provided stable results throughout the considered base and sensitivity periods.

8. Concluding remarks

In this paper, we have proposed and empirically verified an algorithm for the optimization of a simulation execution in the Amazon EC2 spot pricing environment.

We started with the description of the technical characteristics of the Amazon EC2 spot pricing mechanism and proposed the Amazon price simulator implemented in Python and utilizes Amazon's `boto` library for connecting with the EC2 infrastructure. The simulator can be utilized for testing various bidding strategies against historical data.

We ensured comparability of various machine types by performing calculations on the basis of per-ECU costs. For analyses of bidding strategies we have introduced a reservation price constructed as a multiple of the minimum price per ECU-hour.

Simulation results show that bidding just above the spot price is a strategy that produces very good results – both for short as well as for long simulations with check-pointing. This strategy is not very sensitive to the reservation price level across different server types and groups as long as the reservation price does not significantly exceed the on-demand price. For different reservation price levels, the costs does not exceed the minimum reference level by more than 5% (the minimum reference price level is around 17% of the on-demand price).

Simulation results also show that utilizing the free lunch mechanism can lead to a further cost reduction up to around 10% below the minimum reference price, when the bidding is made upon the spot price reaching the minimum level. However, such a strategy significantly increases the time required to complete the simulation (assuming a fixed limit of available EC2 instances) – the effective simulation time is around 33% of simulation time that can be obtained by paying at the minimum reference price level.

We have developed and proposed a specific algorithm for cost optimization in Amazon EC2 spot pricing – the ideas, however, are general and could be applied to other settings. The proposed EC2 spot billing simulation library can also used to test different bidding algorithms that are more suitable to specific computational projects. One possible extension is adaptive killing of a running simulation when we notice that the price for some other market has dropped sharply in comparison to the price of the currently running instance or when the actual performance of a node does not attain the ECU performance level claimed by Amazon.

The bidding algorithm proposed in the paper can be easily integrated with tools for running HPC computations in the Amazon EC2 cloud. For example, the Elastic Load Balancer [31] that is currently a part of Starcluster package [32] supports spot instances and heterogeneous cluster nodes. Its decision-making model can be extended in order to optimize costs while adding new instances to a cluster.

Another interesting area for further research could be simulating the market fundamentals that take place on the Amazon side. Cloud simulation packages such as CloudSim [33] struggle to emulate the behavior of the actual hardware that is used to run a computing cloud. Hence, a multi-agent simulation approach (e.g. similar to the presented in [34]) could be considered for spot cloud modeling. However, Amazon has published very little information regarding construction of the spot market (e.g. no exact information on number or structure of spot instances is known).

An important limitation of our work is related to the fact that Amazon periodically modifies its billing policy and prices. This might render our specific conclusions regarding which specific bidding strategies are most effective invalid after longer periods. Nevertheless, the methodology we have proposed can be applied under these new market conditions and it will allow the identification of new optimal strategies. Yet another research field is extending the proposed adaptive approach

(Algorithm 1) to include other markets when searching for an optimal computing offer. This would require simulating resources and costs required to move simulation between different cloud providers.

Acknowledgments

We would like to thank the three anonymous reviewers for their insightful comments that helped us to improve the article.

References

[1] Scientific computing using spot instances, <http://aws.amazon.com/ec2/purchasing-options/spot-instances/spot-and-science/> (accessed: 30.06.14).
[2] M. Mao, M. Humphrey, A performance study on the vm startup time in the cloud, in: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 423–430.
[3] H. Xu, B. Li, Maximizing revenue with dynamic cloud pricing: the infinite horizon case, in: 2012 IEEE International Conference on Communications (ICC), IEEE, 2012, pp. 2929–2933.
[4] K. Vanmechelen, W. Depoorter, J. Broeckhove, Combining futures and spot markets: a hybrid market approach to economic grid resource management, J. Grid Comput. 9 (1) (2011) 81–94.
[5] C. Vecchiola, S. Pandey, R. Buyya, High-performance cloud computing: a view of scientific applications, in: 2009 10th International Symposium on Pervasive Systems, Algorithms, and Networks (ISPAN), IEEE, 2009, pp. 4–16.
[6] M. Mattess, C. Vecchiola, R. Buyya, Managing peak loads by leasing cloud infrastructure services from a spot market, in: 2010 12th IEEE International Conference on High Performance Computing and Communications (HPCC), IEEE, 2010, pp. 180–188.
[7] B. Javadi, R.K. Thulasiram, R. Buyya, Characterizing spot price dynamics in public cloud environments, Future Gener. Comput. Syst. 29 (4) (2013) 988–999.
[8] A. Andrzejak, D. Kondo, S. Yi, Decision model for cloud computing under sla constraints, in: 2010 IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2010, pp. 257–266.
[9] S. Yi, D. Kondo, A. Andrzejak, Reducing costs of spot instances via checkpointing in the amazon elastic compute cloud, in: 2010 IEEE 3rd International Conference on Cloud Computing (CLOUD), IEEE, 2010, pp. 236–243.
[10] S. Tang, J. Yuan, X.-Y. Li, Towards optimal bidding strategy for amazon ec2 cloud spot instance, in: 2012 IEEE 5th International Conference on Cloud Computing (CLOUD), IEEE, 2012, pp. 91–98.
[11] V. Kushwaha, Y. Simmhan, Cloudy with a spot of opportunity: analysis of spot-priced VMs for practical job scheduling, in: IEEE Cloud Computing for Emerging Markets Conference, Bangalore, 2014.
[12] S.V. Angiuoli, J.R. White, M. Matalka, O. White, W.F. Fricke, Resources and costs for microbial sequence analysis evaluated using virtual machines and cloud computing, PLoS One 6 (10) (2011) e26624.
[13] S. Chaisiri, R. Kaewpuang, B.-S. Lee, D. Niyato, Cost minimization for provisioning virtual servers in amazon elastic compute cloud, in: 2011 IEEE 19th International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), IEEE, 2011, pp. 85–95.
[14] J. Barr, EC2 Spot Instance Termination Notices, <https://aws.amazon.com/blogs/aws/new-ec2-spot-instance-termination-notices/> (accessed 18.01.15).
[15] S. Ostermann, R. Prodan, Impact of variable priced cloud resources on scientific workflow scheduling, in: Euro-Par 2012 Parallel Processing, Springer, Berlin, Heidelberg, 2012, pp. 350–362.
[16] A. Iosup, S. Ostermann, M. Yigitbasi, R. Prodan, T. Fahringer, D. Epema, Performance analysis of cloud computing services for many-tasks scientific computing, IEEE Trans. Parallel Distrib. Syst. 22 (6) (2011) 931–945.
[17] Y. El-Khamra, H. Kim, S. Jha, M. Parashar, Exploring the performance fluctuations of hpc workloads on clouds, in: 2010 IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom), IEEE, 2010, pp. 383–387.
[18] B. Javadi, R. Thulasiram, R. Buyya, Statistical modeling of spot instance prices in public cloud environments, in: Fourth IEEE International Conference on Utility and Cloud Computing (UCC), IEEE, 2011, pp. 219–228.
[19] O. Agmon Ben-Yehuda, M. Ben-Yehuda, A. Schuster, D. Tsafrir, Deconstructing Amazon EC2 spot instance pricing, ACM Trans. Econ. Comput. 1 (3) (2013) 16.
[20] M. Mihailescu, Y. Teo, Dynamic resource pricing on federated clouds, in: 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid), IEEE, 2010, pp. 513–517.
[21] L. Gulyás, A. Szabó, R. Legéndi, T. Máhr, R. Bocsi, G. Kampis, Tools for large scale (distributed) agent-based computational experiments, in: Proceedings of CSSSA 2011, 2011.
[22] H. Liu, Cutting mapreduce cost with spot market, in: 3rd USENIX Workshop on Hot Topics in Cloud Computing, 2011, pp. 1–5.
[23] AWS Price Reduction #42 EC2, S3, RDS, ElastiCache, and Elastic MapReduce, Amazon. <https://aws.amazon.com/blogs/aws/aws-price-reduction-42-ec2-s3-rds-elasticache-and-elastic-mapreduce/> (accessed 28.04.15).
[24] J. Schad, J. Dittrich, J. Quiané-Ruiz, Runtime measurements in the cloud: observing, analyzing, and reducing variance, Proc. VLDB Endowment 3 (1–2) (2010) 460–471.
[25] J. O'Loughlin, L. Gillam, Performance evaluation for cost-efficient public infrastructure cloud use, in: Economics of Grids, Clouds, Systems, and Services, Springer, 2014, pp. 133–145.
[26] I. Raicu, Z. Zhang, M. Wilde, I. Foster, P. Beckman, K. Iskra, B. Clifford, Toward loosely coupled programming on petascale systems, in: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, IEEE Press, 2008, p. 22.
[27] L. Breiman, Random forests, Machine Learn. 45 (1) (2001) 5–32.
[28] L. Breiman, Manual on setting up, using, and understanding random forests v3.1, 2002. <http://oz.berkeley.edu/users/breiman/Using_random_forests_V3.1.pdf>.
[29] A. Liaw, M. Wiener, Classification and regression by randomForest, R News 2 (3) (2002) 18–22.
[30] R Core Team, R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2014. <http://www.R-project.org/>.
[31] R. Banerjee, Elastic Load Balancing in Amazon Compute Cloud, Harvard University, Master thesis, 2011.
[32] StarCluster—open source cluster-computing toolkit for Amazon's Elastic Compute Cloud (EC2), MIT. <http://star.mit.edu/cluster/> (accessed 15.01.15).
[33] R. Calheiros, R. Ranjan, A. Beloglazov, C. De Rose, R. Buyya, CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms, Softw.: Pract. Exp. 41 (1) (2011) 23–50.
[34] J. Werner, G. Geronimo, C. Westphall, F. Koch, R. Freitas, Simulator improvements to validate the green cloud computing approach, in: 7th Latin American on Network Operations and Management Symposium (LANOMS), IEEE, 2011, pp. 1–8.